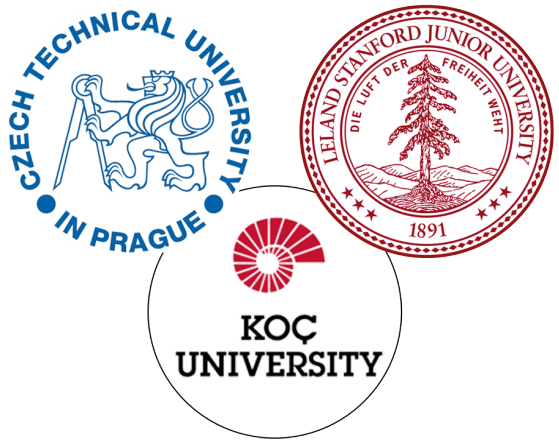
# CS Bridge, Lecture 12

## The Mouse



# Learning Goals

- Learn to respond to mouse events in graphics programs

# Plan for Today

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Plan for Today

- **Recap: Lists**
- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Lists

- A **list** is way to keep track of an *ordered collection* of items
  - Items in the list are called "elements"
  - Ordered: can refer to elements by their position
  - Collection: list can contain multiple items
- The list dynamically adjusts its size as elements are added or removed
- Lists have a lot of built-in functionality to make using them more straightforward

# Lists

- **len(list)** – get the length of a list
- **list.append(elem)** – add elem to the end
- **list[i]** - get ith element
- **list[i] = elem** – set ith element to elem
- **list.insert(i, elem)** – insert elem at ith index
- **list.remove(elem)** – remove first occurrence of elem
- **list.pop(i)** – get and remove ith elem

# Looping Through Elements

```
str_list = ['Leia', 'Luke', 'Han’]

for i in range(len(str_list)):
    elem = str_list[i]
    ...
```

# Looping Through Elements

```
str_list = ['Leia', 'Luke', 'Han']

for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)
```

Output:

```
Leia
Luke
Han
```

# Looping Through Elements

```
str_list = ['Leia', 'Luke', 'Han']

for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)

-----

for elem in str_list:
    print(elem)
```

Output:

```
Leia
Luke
Han
```

# Looping Through Elements

```
str_list = ['Leia', 'Luke', 'Han']

for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)

-----

for elem in str_list:
    print(elem)
    # no i variable here to use
```

Output:

```
Leia
Luke
Han
```

# Looping Through Elements

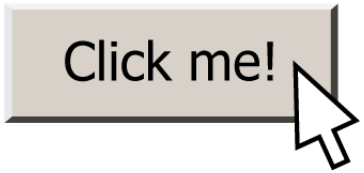
```
str_list = ['Leia', 'Luke', 'Han']

for i in range(len(str_list)):
    elem = str_list[i]
    print(elem)
    if i + 1 < len(str_list):
        # do something with str_list[i + 1]
```

# Plan for Today

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Responding To The Mouse

- **event**: Some external stimulus that your program can respond to.



# Events

- Mouse clicking
- Keyboard keys pressed
- Etc.

# Events

- In our programs, we can ask the canvas if any events have occurred since the last time we asked.
- If there are, then we do something.
- If there are not, we do nothing and check again later.

```
while True:
      # Handle any new mouse events
      # ...
      canvas.update()
```

# Plan for Today

- **Mouse Location**
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Mouse Location

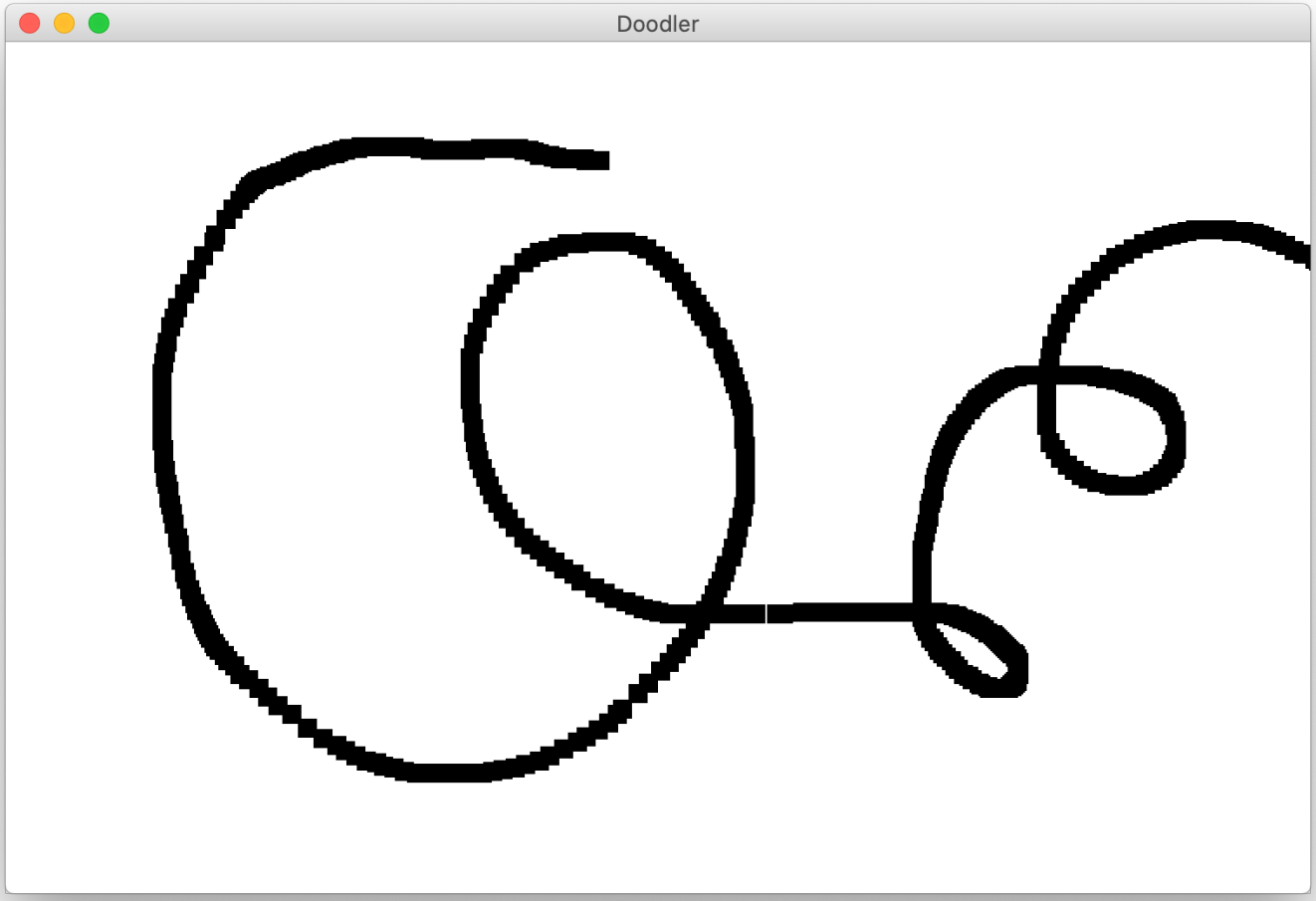
At any time, we can ask the canvas for the current location of the mouse.

```
mouse_x = canvas.get_mouse_x()
mouse_y = canvas.get_mouse_y()
```

# Plan for Today

- Mouse Location
- ***Demo:* Doodler**
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Doodler



# Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')

    canvas.update()
```

# Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')

    canvas.update()
```

# Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')

    canvas.update()
```

# Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')

    canvas.update()
```

# Doodler

```
SQUARE_SIZE = 10
...

while True:
    # Get the mouse location
    mouse_x = canvas.get_mouse_x()
    mouse_y = canvas.get_mouse_y()

    # Create a black rectangle at this location
    rect = canvas.create_rectangle(mouse_x, mouse_y,
                                   mouse_x + SQUARE_SIZE,
                                   mouse_y + SQUARE_SIZE)
    canvas.set_color(rect, 'black')

    canvas.update()
```

# Plan for Today

- Mouse Location
- *Demo:* Doodler
- **Mouse Clicks**
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Mouse Clicks

At any time, we can ask the canvas for a list of mouse clicks that have happened since the last time we asked.

```
clicks = canvas.get_new_mouse_clicks()
```

# Mouse Clicks

Each element in the list has an **x** and **y** coordinate of where that click happened.

```
clicks = canvas.get_new_mouse_clicks()
for click in clicks:
    print(click.x, click.y)
```

# Events

Pattern: we make a loop (like for animation), and each time through the loop we check for new mouse clicks, and act on them.

```
while True:
      # Handle any new mouse clicks
      # ...
      canvas.update()
```

# Plan for Today

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- ***Demo:* Polka Dots**
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# Example: Polka Dots

```
while True:
    clicks = canvas.get_new_mouse_clicks()

    # Add a circle each time the user clicks
    for click in clicks:
        circle = canvas.create_oval(0, 0,
                                    CIRCLE_SIZE, CIRCLE_SIZE)
        canvas.set_color(circle, 'blue')
    canvas.update()
```

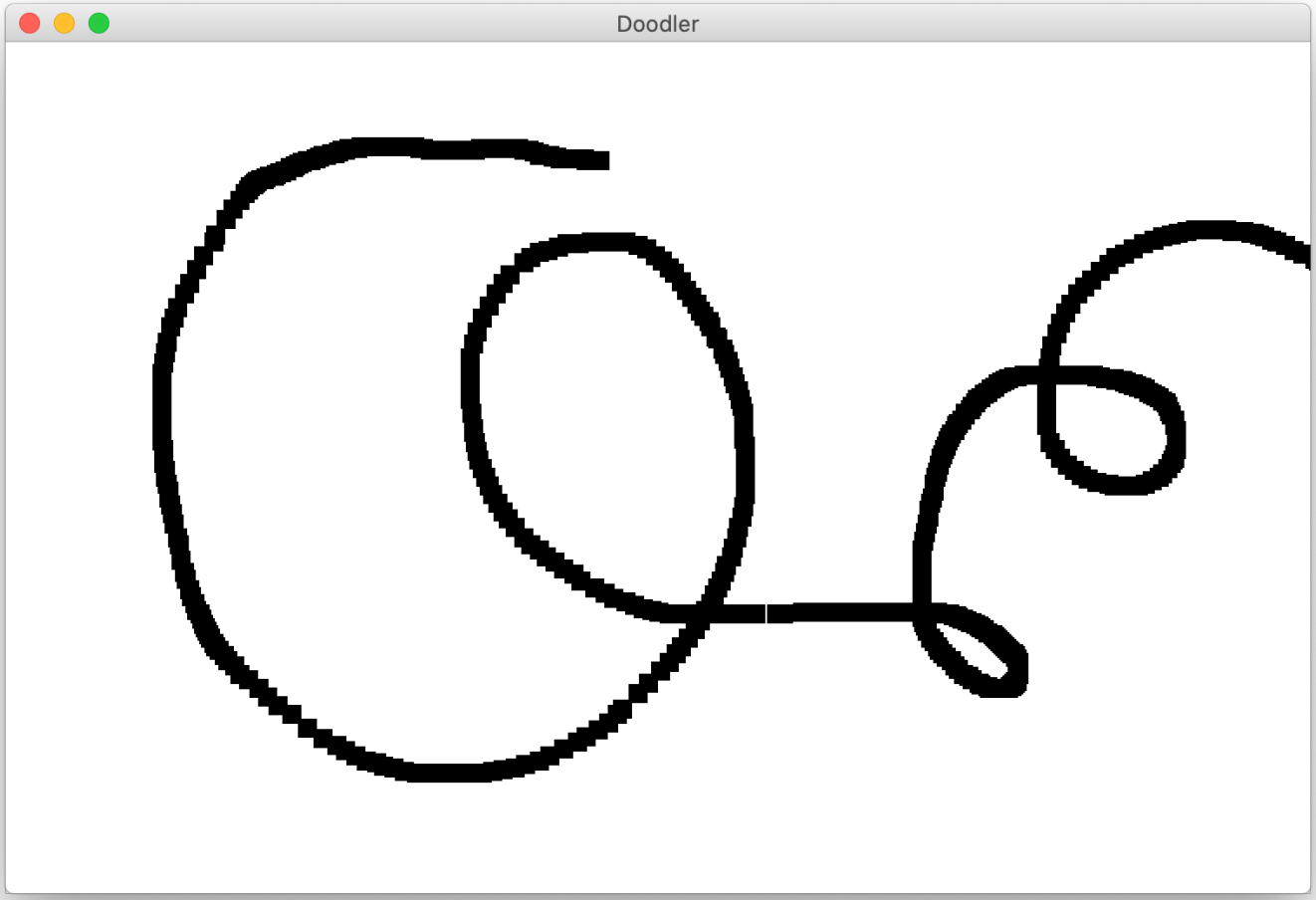
# Example: Polka Dots 2

```
while True:
    clicks = canvas.get_new_mouse_clicks()

    # Add a circle each time the user clicks
    for click in clicks:
        circle = canvas.create_oval(click.x, click.y,
                                    click.x + CIRCLE_SIZE,
                                    click.y + CIRCLE_SIZE)
        canvas.set_color(circle, 'blue')
    canvas.update()
```

# Example: Polka Dots 2

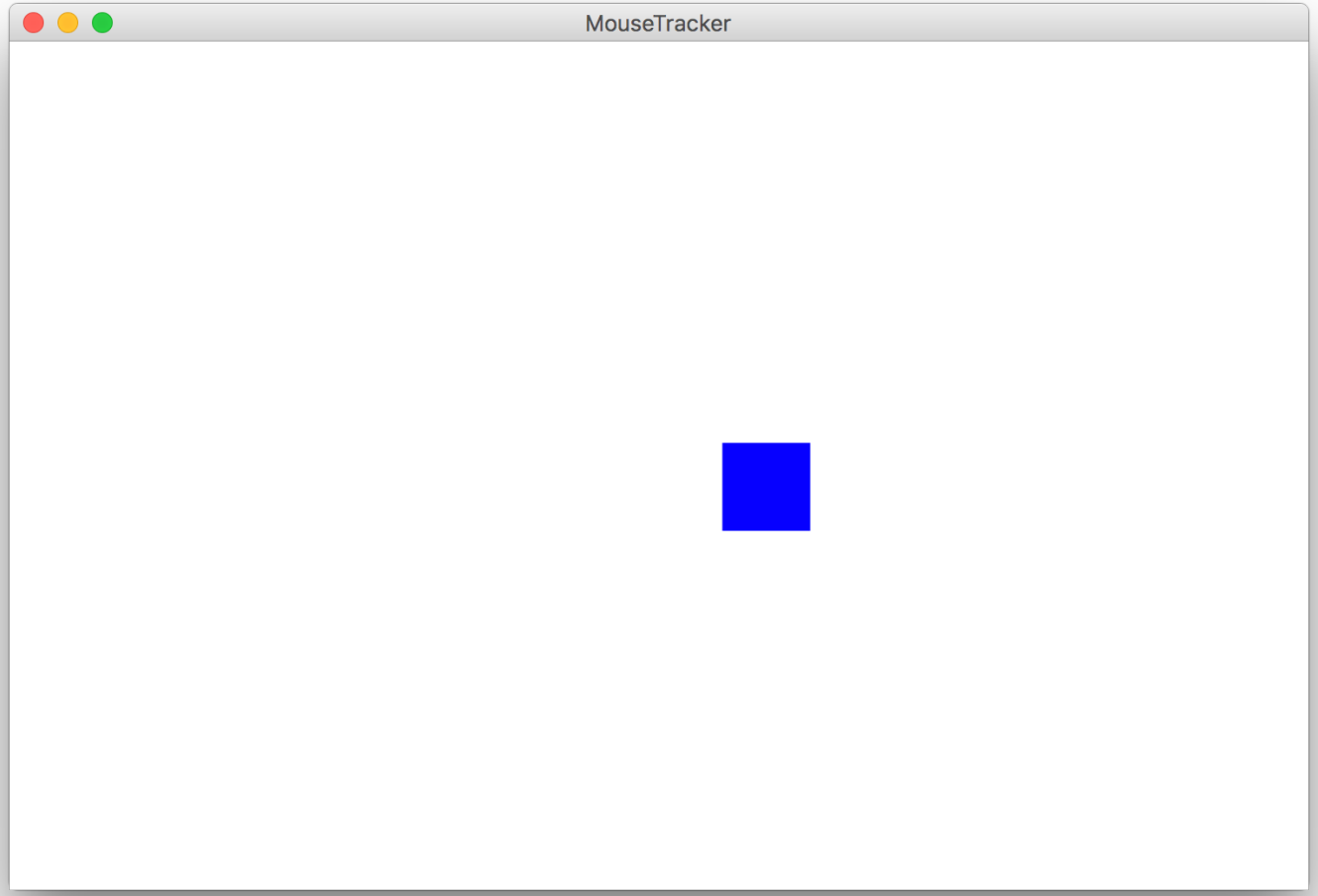
```
while True:
    clicks = canvas.get_new_mouse_clicks()

    # Add a circle each time the user clicks
    for click in clicks:
        circle = canvas.create_oval(click.x, click.y,
                                    click.x + CIRCLE_SIZE,
                                    click.y + CIRCLE_SIZE)
        canvas.set_color(circle, 'blue')
    canvas.update()
```

# Revisiting Doodler




What if we wanted the *same* square to track the mouse, instead of making a new one each time?

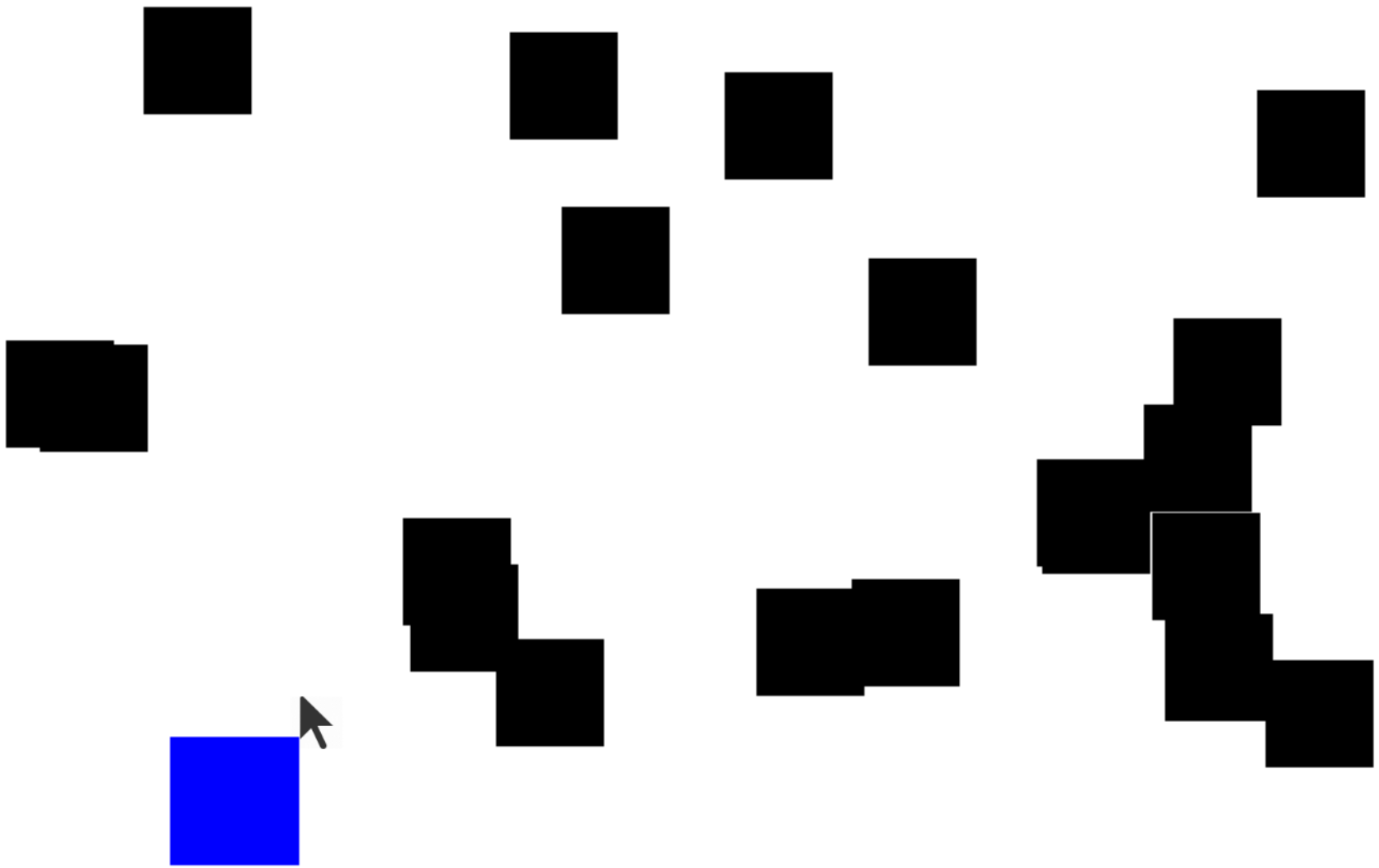
# Example: Mouse Tracker



# Plan for Today

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole

# find_element_at



# find_element_at

find_element_at returns the object at this location on the canvas.

```
object_here = canvas.find_element_at(x, y)
```

# find_element_at

find_element_at returns the object at this location on the canvas.

```
object_here = canvas.find_element_at(x, y)
if object_here:
      // do something with object_here
else:
      // nothing at that location
```

# Putting it all together

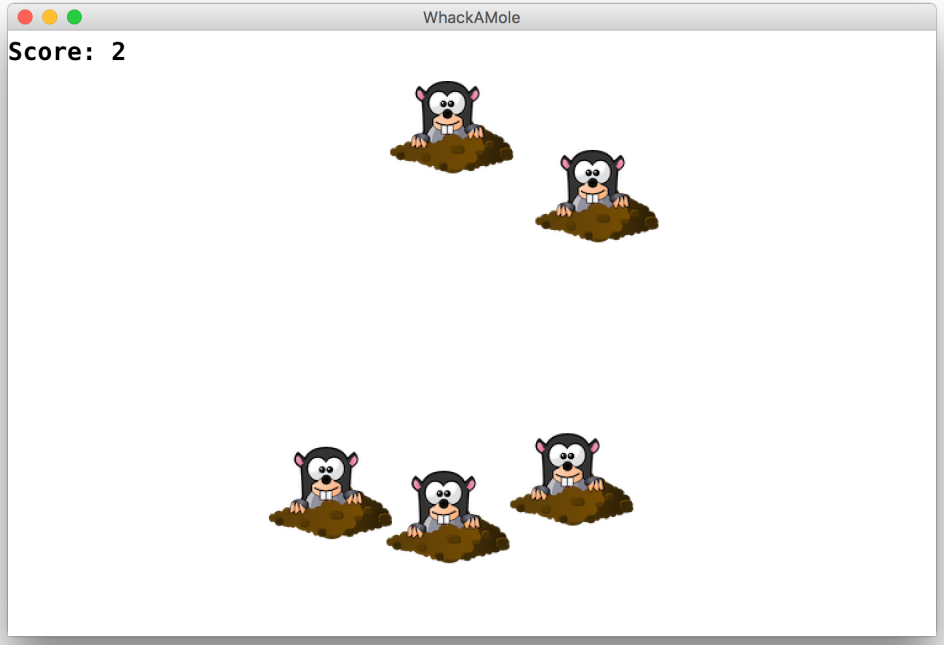
# Plan for Today

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- ***Demo:* Whack-a-Mole**

# Whack-A-Mole

Let's make Whack-A-Mole!

- Moles should appear at random locations on the screen over time
- If the user clicks a mole, remove it



# Recap

- Mouse Location
- *Demo:* Doodler
- Mouse Clicks
- *Demo:* Polka Dots
- **`find_element_at`**
- *Demo:* Whack-a-Mole